# On Approximate Uncomputability of the Kolmogorov Complexity Function*

Ruslan Ishkuvatov[1,2] and Daniil Musatov[1,3,4]

[1] Moscow Institute of Physics and Technology, Russia
[2] LIRMM CNRS & University of Montpellier
[3] Russian Presidential Academy of National Economy and Public Administration, Moscow, Russia
[4] Caucasus Mathematical Center at Adyghe State University, Maykop, Russia
musatych@gmail.com

**Abstract.** Kolmogorov complexity $C(x)$ of a string $x$ is the length of its shortest possible description. It is well known that $C(x)$ is not computable. Moreover, any computable lower estimate of $C(x)$ is bounded by a constant. We study the following question: suppose that we want to compute $C$ with some precision and some amount of errors. For which parameters is it possible? Our main result is the following: the error must be at least an inverse exponential function of the precision. It gives two striking implications. Firstly, no computable function approximate Kolmogorov complexity much better than the length function does. Secondly, time-bounded Kolmogorov complexity is sufficiently far from unbounded Kolmogorov complexity for any particular computable time bound.

## 1 Introduction

Kolmogorov complexity $C(x)$ of a string $x$ is the minimal possible length of a program that generates $x$ for some universal programming language. This notion was introduced in the 1960s and since then the area was comprehensively studied. There are many applications in computability theory, computational complexity, machine learning, statistics etc. Extensive expositions of the subject may be found in books by Li and Vitányi [6] and by Shen, Uspensky and Vereshchagin [8].

A simple observation similar to the Berry paradox shows that Kolmogorov complexity is an uncomputable function. Moreover, there are no computable and unbounded lower estimates. In this paper we study how far is it from being computable. Firstly, we consider two well-known notions of being close to computable. The first one is generic computability: there exists a computable function that is defined almost everywhere and equals to our function on its domain. The second one is coarse computability: there exists a totally computable function that coincides with ours almost everywhere. Then we relax the second

* The reported study was funded by RFBR according to the research project 18-31-00428.

notion by allowing a computable function to be close to the complexity function, not necessarily be equal.

The length function is a good approximation for a majority of strings. Specifically, there are two well-known results:

**Lemma 1.** *There is some constant $c$ such that for any string $x$ of length $n$ it holds that $C(x) \leq n + c$.*

**Lemma 2.** *For any constant $c$ there are less than $2^{n-c}$ strings of length $n$ and complexity less than $n - c$.*

Thus, for all but a fraction $2^{-c}$ of strings it holds that $|\mathrm{len}(x) - C(x)| \leq c$. Our main question is whether this estimate may be substantially improved by another computable function instead of $\mathrm{len}(x)$. The main (and surprising) result is that the answer is negative: the fraction $2^{-c}$ may be replaced by a smaller constant, but it is still a constant for any constant $c$. Moreover, this exponential dependency is preserved for less accurate precisions, where the difference between a computable function and the complexity function grows superconstantly.

### 1.1 Related Work

Algorithmic properties of the complexity function were studied in many sources. Some basic properties are listed in [8, sect. 1.2], [6, sect. 2.7] and [7, sect. 2.1]. The incomputability of $C(x)$ traces back to the pioneering works by Kolmogorov [4] and Solomonoff [9]. Kummer [5] studies algorithmic properties of the set of random strings (that is, the strings for which the complexity is not less then the length) and, in particular, proves that this set is not *frequency enumerable*. This means that there is no computable function $f$ that gets $k$ strings and guesses for at least one of them whether it is random or not. A similar in spirit result due to Beigel et al [2] tells about complexity function itself: if an algorithm produces a list of numbers that is guaranteed to contain $C(x)$, then the length of this list must be linear. Bauwens et al [1] show that, given a string $x$, it is possible to algorithmically produce a small list of programs, one of which generates $x$ and is optimal up to a constant additive term. Unfortunately, this list is not short enough to make any implication about the complexity itself. Fenner and Fortnow [3] show that it is possible to produce a single optimal program for all strings of length $n$ if the generator receives a piece of advice of length approximately $n$.

### 1.2 Roadmap

The rest of the text is organized as follows. In Sect. 2 we strictly define the notions we use and state some basic properties. In Sect. 3 we show that there is no coarsely computable function that approximates $C(x)$ with a constant precision. In Sect. 4 we prove our main result about the exact relationship between the precision of approximation and the number of errors. In Sect. 5 we give a brief conclusion and present some open questions.

## 2 Preliminaries

In this section we give precise definitions of the notions we use.

### 2.1 Kolmogorov Complexity

A simple definition "Kolmogorov complexity of a string is the length of a shortest program that generates this string" lacks the specification of a programming language. To address this issue, we consider the notion of a description method, or a decompressor. We define a more general notion of conditional complexity.

**Definition 1.** *Let $D$ be a computable function with two arguments. The complexity $C_D(x|y)$ of a string $x \in \{0,1\}^*$ conditional on a string $y \in \{0,1\}^*$ with respect to a decompressor $D$ is the minimal length of a string $p$ such that $D(p,y) = x$. If there is no such $p$, then $C_D(x|y) = \infty$.*

For any particular string $x$, one may consider a decompressor that hardwires $x$ and outputs it, say, on the empty program $p$. Thus, changing the decompressor may drastically change the complexity. The following celebrated theorem shows that this change is limited.

**Theorem 1 (Kolmogorov-Solomonoff, [4, 9]).** *There exists a decompressor $U$ such that for any decompressor $D$ there exists a constant $c$ such that for all $x$ and $y$ it holds that $C_U(x|y) \leq C_D(x|y) + c$.*

Such machine is called a universal decompressor. The proof idea is simple: $U$ treats a part of its first input as a description of $D$ and launches it on the rest of the input. Usually a particular $U$ is fixed and the index is omitted. Most equations are valid up to some additive constant. In the paper we consider two specific complexities:

**Definition 2.** *Let $x$ be some string. Its* unconditional complexity *$C(x)$ is just $C(x|\epsilon)$, where $\epsilon$ is the empty string. The* length conditional complexity *of $x$ is $C(x|n)$, where $n = |x|$.*

The idea behind length conditional complexity is the following. Complexity may be considered as a measure of information contained in a particular string. If a string is a prefix of a computable sequence, then its complexity equals the complexity of its length plus some constant. If a string is random, then its complexity is close to its length. Length condition makes the complexity of a computable string essentially zero, but keeps the complexity of a random string close to its length. Thus length condition helps to separate the information about the length of the string from the information in the string itself.

We consider also a time-bounded version of Kolmogorov complexity.

**Definition 3.** *Let $D$ be a Turing machine. The complexity $C^t_D(x|y)$ of a string $x \in \{0,1\}^*$ conditional on $y$ in time $t$ with respect to decompressor $D$ is the minimal length of a string $p$ such that $D(p,y) = x$ and $D(p,y)$ halts in at most $t$ steps. If there are no such $p$, then $C^t_D(x|y) = \infty$.*

For the time-bounded version the universal decompression theorem is the following:

**Theorem 2.** *There exists a decompressor $U$ such that for any decompressor $D$ there exist constants $c$ and $d$ such that for all $x$, $y$ and $t$ it holds that $C_U^{dt\log t}(x|y) \leq C_D^t(x|y) + c$.*

It is clear that a universal decompressor in the time-bounded framework is also a universal decompressor in the unbounded framework. In the sequel we fix some such decompressor, and thus obtain $C^t(x|y) \geq C(x|y)$ for all $x$, $y$ and $t$ without adding a constant.

### 2.2 Generic and Coarse Computability

Apart from computable functions, one may consider functions that are computable "almost everywhere". This notion may be formalized in two non-equivalent ways. Both of them employ the notion of asymptotic density.

**Definition 4.** *Let $S \subset \{0,1\}^*$. The density of $S$ in length $n$ is defined as the fraction of length-$n$ strings that lie in $S$, i.e., $\rho_n(S) = \frac{|S\cap\{0,1\}^n|}{2^n}$. If the sequence $\rho_n(S)$ has a limit $\rho(S)$, then it is called the* asymptotic density *of $S$. If $\rho(S) = 1$, we call $S$ a* generic set*. If $\rho(S) = 0$, we call $S$ a* negligible set*.*

**Definition 5.** *A total function $h\colon \{0,1\}^* \to \{0,1\}^*$ is called* generic computable *if there exists a partially computable function $f\colon \{0,1\}^* \to \{0,1\}^*$ such that if $f(x)$ is defined, then $f(x) = h(x)$, and the domain of $f$ is a generic set.*

**Definition 6.** *A total function $h\colon \{0,1\}^* \to \{0,1\}^*$ is called* coarsely computable *if there exists a total computable function $f\colon \{0,1\}^* \to \{0,1\}^*$ such that the set $\{x \mid f(x) = h(x)\}$ is a generic set.*

## 3 Approximating Kolmogorov Complexity with Constant Accuracy

The following theorem shows that there does not exist generic computable lower estimate of $C(x)$ in any sense.

**Theorem 3.** *There is no partially computable function $f(x)$ such that $f(x) \leq C(x)$ in the domain of $f$ and $f(x)$ takes on arbitrarily large values.*

*Proof.* The standard argument is valid here. Suppose that such function $f$ exists. From its definition, the set $\{x \mid f(x) > n\}$ is non-empty and computably enumerable. The complexity of the first element $x_0$ in an enumeration is at most $\log n + O(1)$. On the other hand, $C(x_0) \geq f(x_0) > n$, hence a contradiction for a sufficiently large $n$.

On the other hand, $C(x)$ does have coarsely computable lower estimates. For instance, $f(x) = \min\left\{\frac{|x|}{2}, C(x)\right\}$ is not greater than $C(x)$ and coincides with computable function $\frac{|x|}{2}$ with asymptotic density 1. So, the question arises how accurately can $C(x)$ be approximated by a coarsely computable function. It turns out that any computable superconstant accuracy can be achieved: just take $f(x) = \min\{|x| - \alpha(|x|), C(x)\}$ for any computable function $\alpha(\cdot)$ that tends to infinity. Such a function is not greater than $C(x)$ and coincides with $|x| - \alpha(|x|)$ with density $O(2^{-\alpha(|x|)})$ that tends to zero. We want to show that a constant accuracy cannot be achieved. We start from showing a similar fact about the length conditional complexity $C(x|n)$.

**Theorem 4.** *There is no coarsely computable function $f(x)$ such that for some constant $c$ it holds that $C(x|n) - c \leq f(x) \leq C(x|n)$ for all $x$.*

*Proof.* Suppose that such $f(x)$ exists for some $c$. Consider a computable function $h$ that coincides with $f$ with asymptotic density 1. Take an arbitrary number $\delta = 2^{-k}$, where $k > 2$. From the definition, there must exist $N$ such that for all $n > N$ the fraction of strings $x$ of length $n$ with different values $f(x)$ and $h(x)$ is less than $\delta$. Now fix an arbitrary $n > N$. Consider a program $P$ that gets $n$ and enumerates all strings of length $n$ in a specific order. The strings are ordered by $h(x)$ in a non-increasing manner, and then lexicographically. Denote by $x_i$ the $i$th string in this enumeration. It is clear that $C(x_i|n) \leq C(i) + O(1)$, where the constant in $O(1)$ depends on the program that computes $h$. From the properties of $h$ and the pigeonhole principle, there must exist some $j \leq 2^{n-k}$ such that $f(x_j) = h(x_j)$. We fix such $x_j$ and estimate its complexity. On the one hand, it has low complexity: $C(x_j|n) \leq C(j) + O(1) \leq n - k + O(1)$. On the other hand, it must have high complexity. Specifically, for at least half of the strings it holds that $C(x_i|n) \geq n - 1$. Since $f(x) \geq C(x|n) - c$, for at least half of the strings it holds that $f(x_i) \geq n - c - 1$. Because $\delta < \frac{1}{4}$, for at least a quarter of the strings it holds simultaneously that $f(x_i) = h(x_i)$ and $f(x_i) \geq n - c - 1$. Since the output of $P$ is ordered by the value of $h$ in a non-increasing way, for all strings in the first quarter of the list it holds that $h(x_i) \geq n-c-1$. Thus, for the previously fixed $j$ it holds that $C(x_j|n) \geq f(x_j) \geq n - c - 1$. Now we obtain a contradiction: on the one hand, $C(x_j|n) \leq n - k + O(1)$. On the other hand, $C(x_j|n) \geq n-c-1$. Since $c$ is fixed and $k$ is arbitrary, we obtain a contradiction for large enough $k$.

This proof cannot be literally reproduced for the unconditional complexity, because a logarithmic term should be added to the upper bound, but not to the lower one, and with this addition the contradiction vanishes. Instead, we employ the following lemma that compares the complexities $C(x)$ and $C(x|n)$.

**Lemma 3.** *Suppose that $C(x|n) < n - k$. Then $C(x) \leq n - k + O(\log k)$.*

*Proof.* Firstly, we modify the description method such that it obtains the following property: if $C(x|n) < n - k$, then there exists a description of length exactly $n - k$. This is done by the following: discard the leading zeros and the first one from a description and then launch a usual decompressor. In this case any number of zeros may be attached from the beginning without changing the output, and thus the desired property is satisfied.

Secondly, the description of $x$ now consists of a description of $k$ and a description of $x$ conditional on $n$ of length exactly $n-k$. By restoring $k$ and adding it to the length of the description we may compute $n$ and then obtain $x$. The total length of the description is $n-k+O(\log k)$, as claimed.

Now we are ready to expand the result to the case of unconditional complexity. It is not a direct corollary of theorem 4, but can be obtained by a similar argument.

**Theorem 5.** *There is no coarsely computable function $f(x)$ such that for some constant $c$ it holds that $C(x)-c \leq f(x) \leq C(x)$ for all $x$.*

*Proof.* The proof proceeds like the proof of theorem 4. We suppose that such $f(x)$ exists, denote by $h(x)$ the respective computable function and define $\delta = 2^{-k}$ as before. Consider a large enough $n$ and the enumeration of all strings of length $n$ ordered by $h(x)$ in the non-increasing manner, and then lexicographically. As before, we denote by $x_j$ the earliest $x$ such that $f(x) = h(x)$. On the one hand, we have $f(x_j) \geq n-c-1$ and thus $C(x_j) \geq n-c-1$. On the other hand, we have $C(x_j|n) \leq n-k+O(1)$, as before. By lemma 3 we get $C(x_j) \leq n-k+O(\log k)$. Since $c$ is fixed and $k$ is arbitrary, the two bounds contradict for large enough $k$.

Note that now we can modify the theorem to be symmetric:

**Corollary 1.** *There is no coarsely computable function $f(x)$ such that for some constant $d$ it holds that $|C(x)-f(x)| \leq d$ for all $x$.*

*Proof.* If such function $f(x)$ exists, then $f(x)-d$ contradicts theorem 5 for $c = 2d$.

One interesting corollary deals with computable upper estimates of $C(x)$.

**Corollary 2.** *Suppose that $t(n)$ is some total computable function. Then for any constant $d$ and all sufficiently large $n$ the density of $x \in \{0,1\}^n$ such that $C^{t(n)}(x) - C(x) \geq d$ is bounded away from zero.*

This is more than a direct application of corollary 1, because we claim not only that the density does not tend to zero but also that it is bounded away from zero. This is why we repeat a part of the proof.

*Proof.* Suppose that, on the contrary, there exists a constant $d$ and an increasing sequence $n_m$, such that the fraction of $x \in \{0,1\}^{n_m}$ such that $C^{t(n_m)}(x) - C(x) \leq d$ tends to zero. We repeat the argument from the proof of theorem 5 for such values of $n$ and $h(x) = C^{t(|x|)}(x)$. All $x \in \{0,1\}^n$ are sorted by $h$ in a non-increasing order, then lexicographically. Among the first $2^{n-k}$ strings there must be $x_j$, such that $h(x_j) \leq C(x_j) + d$. On the one hand, $h(x_j)$ must be at least $n-1$, thus $C(x_j) \geq n-d-1$. On the other hand, $C(x_j|n) \leq n-k$ and a contradiction follows.

This corollary is very meaningful: giving more working time allows to substantially economize on the program length. It does not matter, how much time do you already have—polynomial, exponential, power of exponents etc.—allowing unlimited time may shorten a constant fraction of programs by more than a constant number of bits.

## 4 Measuring the Exact Accuracy

In this section we do a more precise analysis. We consider two parameters: a threshold on the difference between a complexity function and a totally computable function, and the fraction of strings that exceed this difference. Formally, we use the following definition:

**Definition 7.** *Let $F\colon \{0,1\}^* \to \mathbb{N}$ be some function, $d\colon \mathbb{N} \to \mathbb{N}$ be a total computable function, and $\alpha\colon \mathbb{N} \to (0,1)$ be another function. We say that $F$ is* approximately computable *with precision $d(n)$ and error $\alpha(n)$ if there exists a total computable function $h\colon \{0,1\}^* \to \mathbb{N}$ such that for all large enough $n$ the fraction of $x \in \{0,1\}^n$ satisfying $|F(x) - h(x)| > d(n)$ does not exceed $\alpha(n)$. If, moreover, it holds that $h(x) \geq F(x)$ for all $x$ (resp., $h(x) \leq F(x)$), we call $F$* approximately computable from above *(resp., approximately computable from below).*

Note that by modifying $h$ in a finite number of arguments we may replace "for all large enough $n$" by "for all $n$". In these terms theorems 4 and 5 may be restated as follows: for any constant $d \in \mathbb{N}$ and any function $\alpha(n) = o(1)$ the functions $C(x|n)$ and $C(x)$ are not approximately computable with precision $d$ and error $\alpha$. Now we consider the case of a non-constant precision.

### 4.1 Approximating Length Conditional Complexity

In this section we prove a generalized version of theorem 4. We start by considering approximate computability from above. This leads to a corollary about approximation of the plain complexity by the time-bounded complexity.

**Theorem 6.** *Let $d(n) < n - \Omega(1)$ be a total computable function. Then $C(x|n)$ is not approximately computable from above with precision $d(n)$ and error $\alpha(n) = o(2^{-d(n)})$.*

Note that if $d(n) = n - O(1)$, then the theorem is wrong, at least for some choice of the decompressor. Indeed, we can think that $c_1 < C(x|n) < n + c_2$ for arbitrarily large $c_1 - c_2$. In this case $n + c_2$ is a computable upper bound. The length function is also an approximation with an arbitrary precision $d(n)$ and error $O(2^{-d(n)})$, so this bound is tight.

*Proof.* Let $h(x)$ be a computable approximation of $C(x|n)$ from above with precision $d(n)$ and error $\alpha(n) = o(2^{-d(n)})$. Take an arbitrary $\delta = 2^{-k}$ and $n$ so large that $\alpha(n) < \delta \cdot 2^{-d(n)} = 2^{-d(n)-k}$. Consider the enumeration of

$\{0,1\}^n$ by $h(x)$ in a non-decreasing order, then lexicographically. Among the first $2^{n-d(n)-k}$ elements there must exist $x_j$ such that $h(x_j) \leq C(x_j|n) + d(n)$. On the one hand, since $h(x) \geq C(x|n)$ and $x_j$ is in the first half of the list, it must hold that $h(x_j) \geq n-1$, and thus $C(x_j|n) \geq n - d(n) - 1$. On the other hand, $C(x_j|n) \leq n - d(n) - k + O(1)$, because $x_j$ can be specified by its ordinal number in the enumeration. If $k$ is large enough, we obtain a contradiction.

Now consider the general case of approximate computability.

**Theorem 7.** *Let $d(n) < \frac{n}{2} - \Omega(1)$ be a total computable function. Then $C(x|n)$ is not approximately computable with precision $d(n)$ and error $\alpha(n) = o(2^{-2d(n)})$.*

Again the claim is wrong for $d(n) = \frac{n}{2} - O(1)$. The function $\frac{n}{2} + c$ is a good approximation. Error $O(2^{-2d(n)})$ is achieved by a simple approximation $n - d(n)$, so this bound is also tight.

*Proof.* Let $h(x)$ be a computable approximation of $C(x|n)$ with precision $d(n)$ and error $\alpha(n) = o(2^{-2d(n)})$. Take an arbitrary $\delta = 2^{-k}$, $k > 2$, and take $n$ so large that $\alpha(n) < \delta \cdot 2^{-2d(n)} = 2^{-2d(n)-k}$. Consider the enumeration of $\{0,1\}^n$ by $h(x)$ in a non-increasing order, then lexicographically. Among the first $2^{n-2d(n)-k}$ elements there must exist $x_j$ such that $|h(x_j) - C(x_j|n)| \leq d(n)$. On the one hand, for at least half of $x$ it holds that $C(x|n) \geq n-1$, thus for at least a quarter of $x$ it holds that $h(x) \geq n - d(n) - 1$. Since $k > 2$, the chosen $x_j$ must lie in this quarter. We obtain $C(x_j|n) \geq h(x_j) - d(n) \geq n - 2d(n) - 1$. On the other hand, $C(x_j|n) \leq n - 2d(n) - k + O(1)$. If $k$ is large enough, we obtain a contradiction.

Thus, two-sided approximation can be obtained with an error smaller than one-sided approximation (note that approximation from below cannot be made at all). But the order of this approximation is the same: for instance, logarithmic precision may be achieved for all but inverse polynomial fraction of strings.

## 4.2 Approximating Plain Complexity

In order to obtain a result about approximating plain complexity $C(x)$, we need to prove an analogue of lemma 3. Direct application produces logarithmic discrepancies and leads to a weaker theorem. We slightly change the statement and employ the fact that $d(n)$ is computable.

**Lemma 4.** *Suppose that $C(x|n) < n-k$. Then $C(x) \leq n-k+C(n|n-k)+O(1)$.*

*Proof.* The proof proceeds along the lines of the proof of lemma 3. At the last step we replace the description of $k$ by a description of $n$ conditional on $n-k$, that is sufficient to restore $n$.

When is the complexity $C(n|n-d(n))$ constant? For instance, if $d(n)$ is growing slow. Specifically, we use the following mild condition.

**Definition 8.** *Let $d\colon \mathbb{N} \to \mathbb{N}$ be a total computable function. Say that $d$ grows* uniformly slower than linearly *if there exists a constant $c$ such that for all $n$ and $m$ such that $m > n + c$ it holds that $d(m) - d(n) < m - n$.*

It is clear that "usual" functions, like logarithms, polylogarithmic functions, power functions $n^\delta$ for $\delta < 1$, linear functions $\alpha n$ for $\alpha < 1$, etc. all possess this property.

**Lemma 5.** *If $d$ is totally computable and grows uniformly slower than linearly, then $C(n|n - d(n)) = O(1)$.*

*Proof.* Let $K$ be some number and $n_0$ be the smallest $n$ such that $n - d(n) = K$. If $m > n_0 + c$, then $m - d(m) > n_0 - d(n_0)$ from the properties of $d$. Thus, any $n$ with $n - d(n) = K$ must lie in $[n_0, n_0 + c]$. Since $c$ is a constant, one needs a constant number of bits to specify a particular $n$.

Now we are ready to prove the theorems.

**Theorem 8.** *Let $d(n) < n - \Omega(1)$ be a total computable function that grows uniformly slower than linearly. Then $C(x)$ is not approximately computable from above with precision $d(n)$ and error $\alpha(n) = o(2^{-d(n)})$.*

**Theorem 9.** *Let $d(n) < \frac{n}{2} - \Omega(1)$ be a total computable function such that $2d(n)$ grows uniformly slower than linearly. Then $C(x)$ is not approximately computable with precision $d(n)$ and error $\alpha(n) = o(2^{-2d(n)})$.*

*Proof.* The proofs proceed along the same lines up to the condition $C(x_j|n) \leq n - d(n) - k + O(1)$ (resp., $C(x_j|n) \leq n - 2d(n) - k + O(1)$). By applying lemma 4, we get $C(x_j) \leq n - d(n) - k + C(n|n - d(n) - k) + O(1) \leq n - d(n) - k + C(n|n - d(n)) + C(k) + O(1) \leq n - d(n) - k + O(\log k)$, where the last inequality employs lemma 5. The contradiction for large enough $k$ still holds.

As before, we obtain a proposition about time-bounded complexity:

**Corollary 3.** *Suppose that $t(n)$ is some computable function. Then for any computable function $d(n) = n - \Omega(1)$ that grows slower then linearly the density of $x$ such that $C^{t(|x|)}(x) - C(x) \geq d(|x|)$ is $\Omega(2^{-d(|x|)})$.*

The proof combines the previously used techniques and thus is omitted. The informal meaning expands that of corollary 2: allowing unlimited time may decrease the program length by at least logarithm for an inverse polynomial fraction of the strings, by at least $n^{-\delta}$ for the fraction at least $\Omega(\frac{1}{2^{n^{-\delta}}})$ etc.

## 5 Conclusion

In this paper we introduced the notion of approximate computability and analyzed it for the case of Kolmogorov complexity function. Despite its naturalness, it seems to have never been appeared in the literature. It would be interesting to study the structural properties of approximately computable functions. For instance, what can be done with an oracle that approximately computes $C(x)$ with sufficiently small precision and error? Can one then compute $C(x)$ exactly? What other functions that are not approximately computable may be constructed? Are there natural examples that do not deal with Kolmogorov complexity? How can these examples be classified? Is there any nice hierarchy?

**Acknowledgments**

We want to thank Alexei Milovanov and Alexander Shen for their support and advice during the work on this paper. We want to thank three anonymous referees for their useful comments about the text and the attendant of a seminar in LIRMM for their attention and questions.

## References

1. Bauwens, B., Makhlin, A., Vereshchagin, N., Zimand, M.: Short lists with short programs in short time. In: Computational Complexity (CCC), 2013 IEEE Conference on. pp. 98–108. IEEE (2013)
2. Beigel, R., Buhrman, H., Fejer, P., Fortnow, L., Grabowski, P., Longpré, L., Muchnik, A., Stephan, F., Torenvliet, L.: Enumerations of the Kolmogorov function. The Journal of Symbolic Logic **71**(2), 501–528 (2006)
3. Fenner, S., Fortnow, L.: Compression complexity. arXiv preprint arXiv:1702.04779 (2017), https://arxiv.org/abs/1702.04779
4. Kolmogorov, A.N.: Three approaches to the quantitative definition ofinformation'. Problems of information transmission **1**(1), 1–7 (1965)
5. Kummer, M.: On the complexity of random strings. In: Annual Symposium on Theoretical Aspects of Computer Science. pp. 25–36. Springer (1996)
6. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and Its Applications, Third Edition. Texts in Computer Science, Springer (2008). https://doi.org/10.1007/978-0-387-49820-1, http://dx.doi.org/10.1007/978-0-387-49820-1
7. Nies, A.: Computability and randomness, vol. 51. OUP Oxford (2009)
8. Shen, A., Uspensky, V.A., Vereshchagin, N.: Kolmogorov complexity and algorithmic randomness, vol. 220. American Mathematical Soc. (2017)
9. Solomonoff, R.J.: A formal theory of inductive inference. part i. Information and control **7**(1), 1–22 (1964)